Building Agentic AI applications on AWS

Vamsi Vikash Ankam

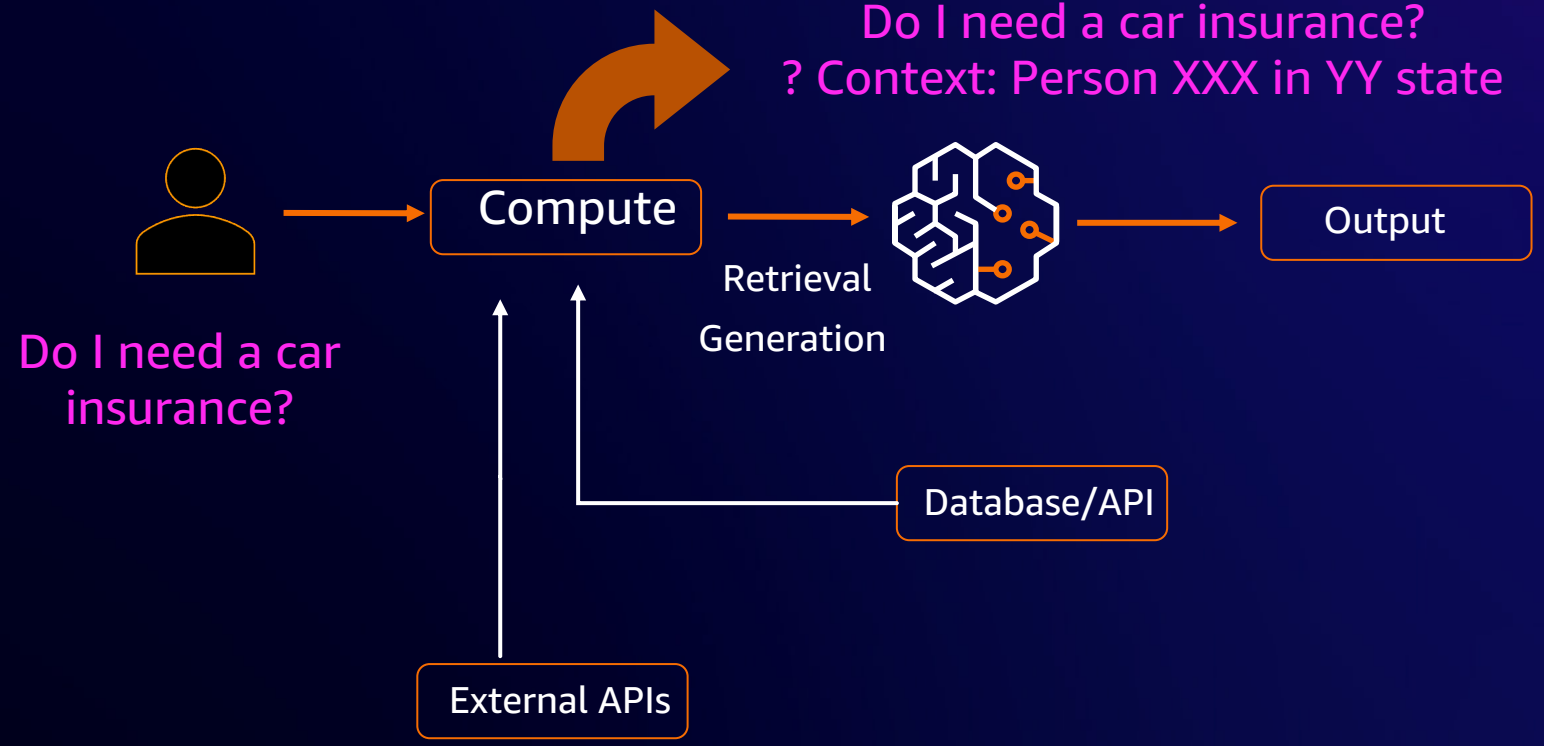
Sr. Serverless Solution Architect

Agenda

- What is an AI Agent?
- Types of AI Agents
- Building Agentic AI applications on AWS
- How MCP helps Agentic AI applications
- **Common patterns
- **Best practices

Rise of Agentic AI Systems – Simple invocation

Rise of Agentic AI Systems – Domain specific invocation



Agentic AI or RAG?

RAG

Retrieve and injects relevant context into LLMs from static, pre-processed data to enhance factual accuracy, typically in a fixed, low-autonomy cycle

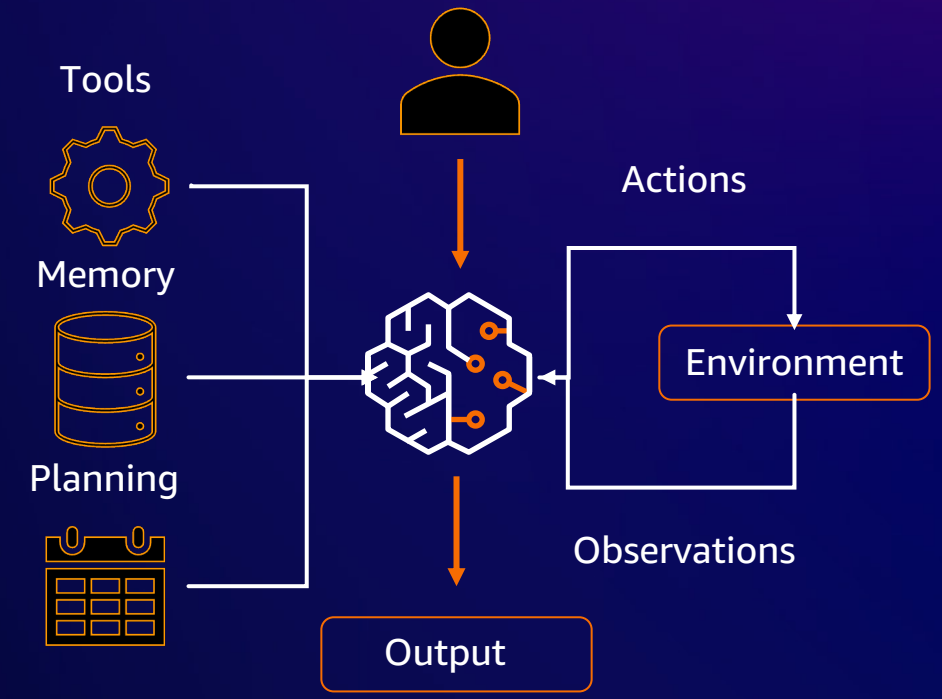
Agentic AI

Use highly-autonomy agents to plan, reason, and act through dynamic, multi-step tasks by leveraging real-time data and **external tools** in adaptive cycles

Rise of Agentic AI Systems – AI agents

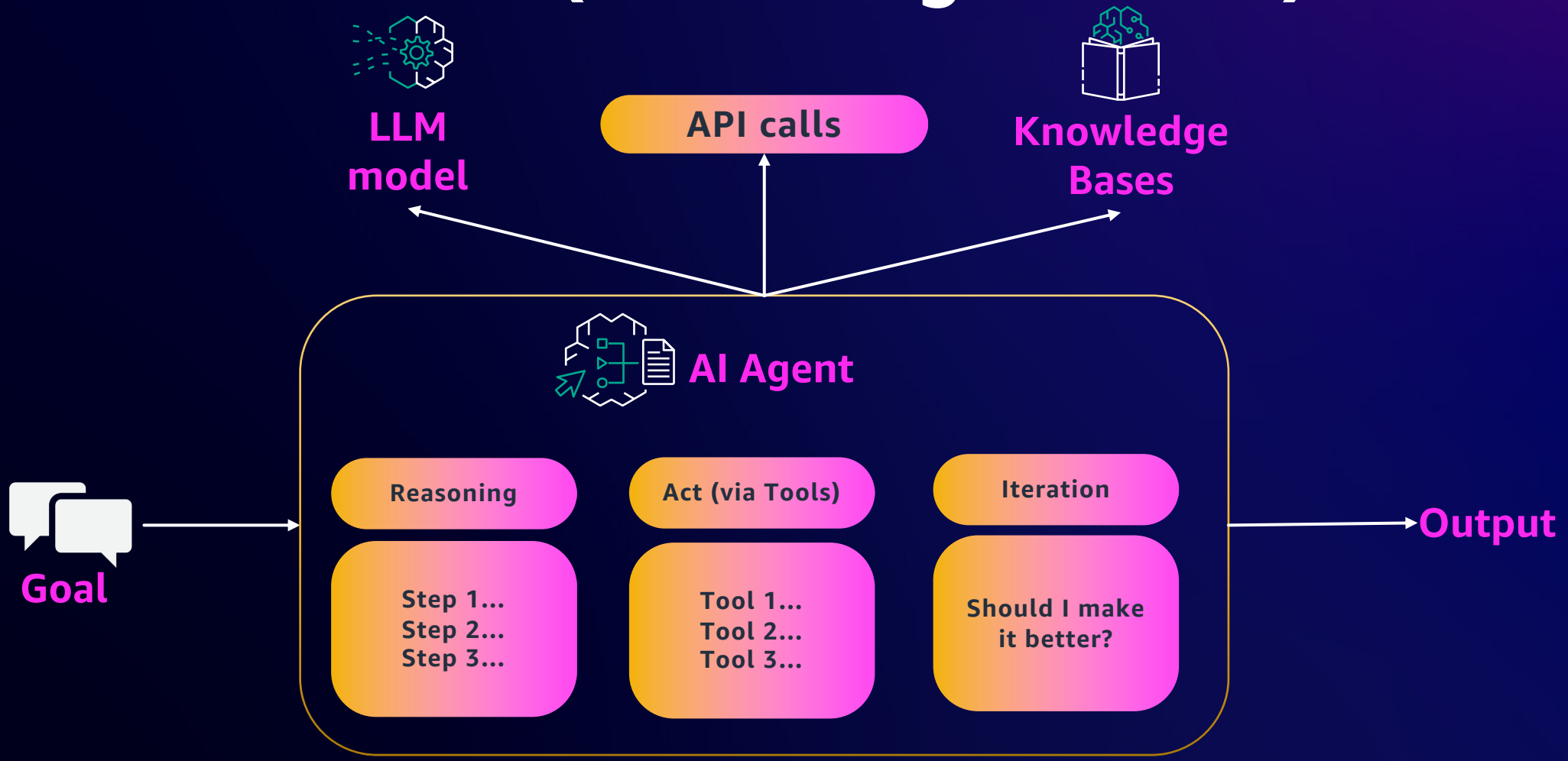
Finance Analyst

How does the Amazon's revenue, profit margins, and cash flow look in 2025? Are there any trends or significant changes year-over-year?

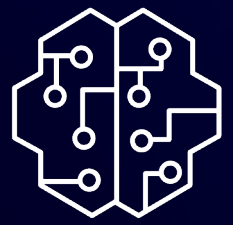


AI Agentic System

ReAct Framework (Reasoning + Action)



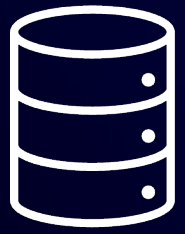
AI Agentic Systems



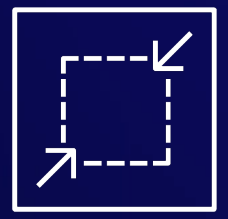
Plan and sequence actions to achieve a goal

Perceive and process context aware information

Use tools to perform tasks efficiently

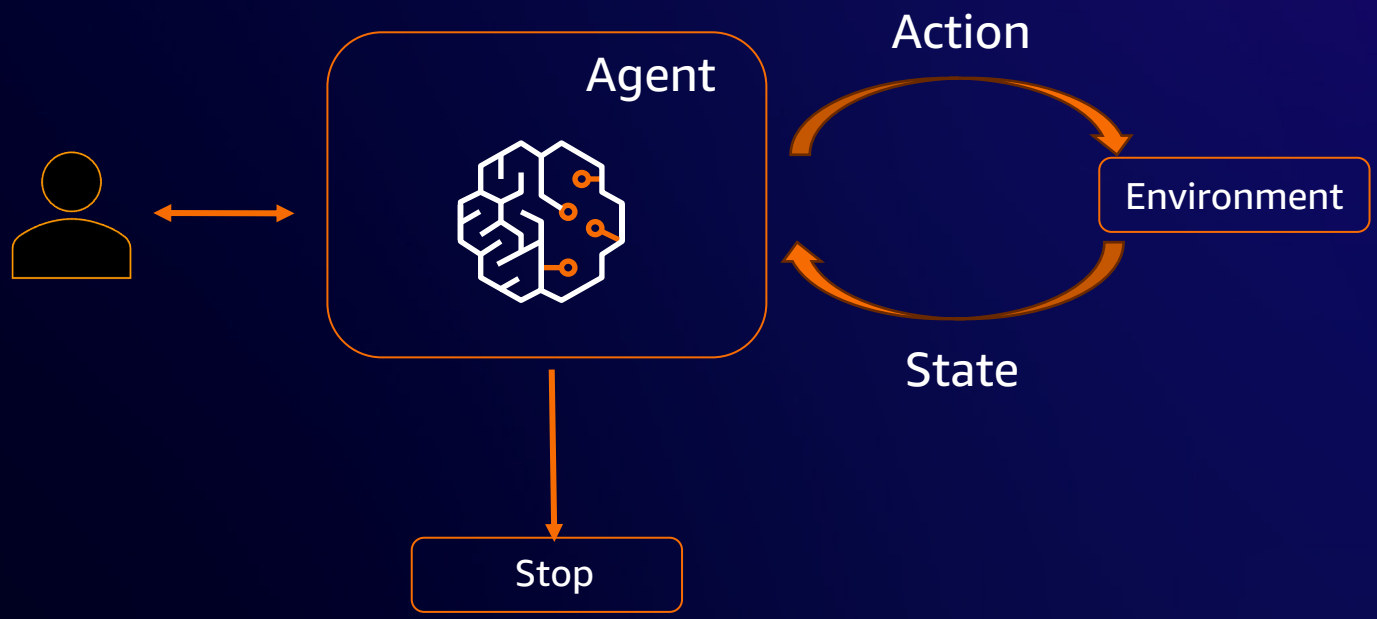


Remember past interactions and behaviors



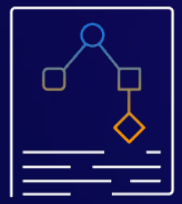
Makes decisions and execute actions based on goals

Agentic systems rely on action and state



Agent is an orchestrator

Decompose into steps using available actions and Knowledge Bases



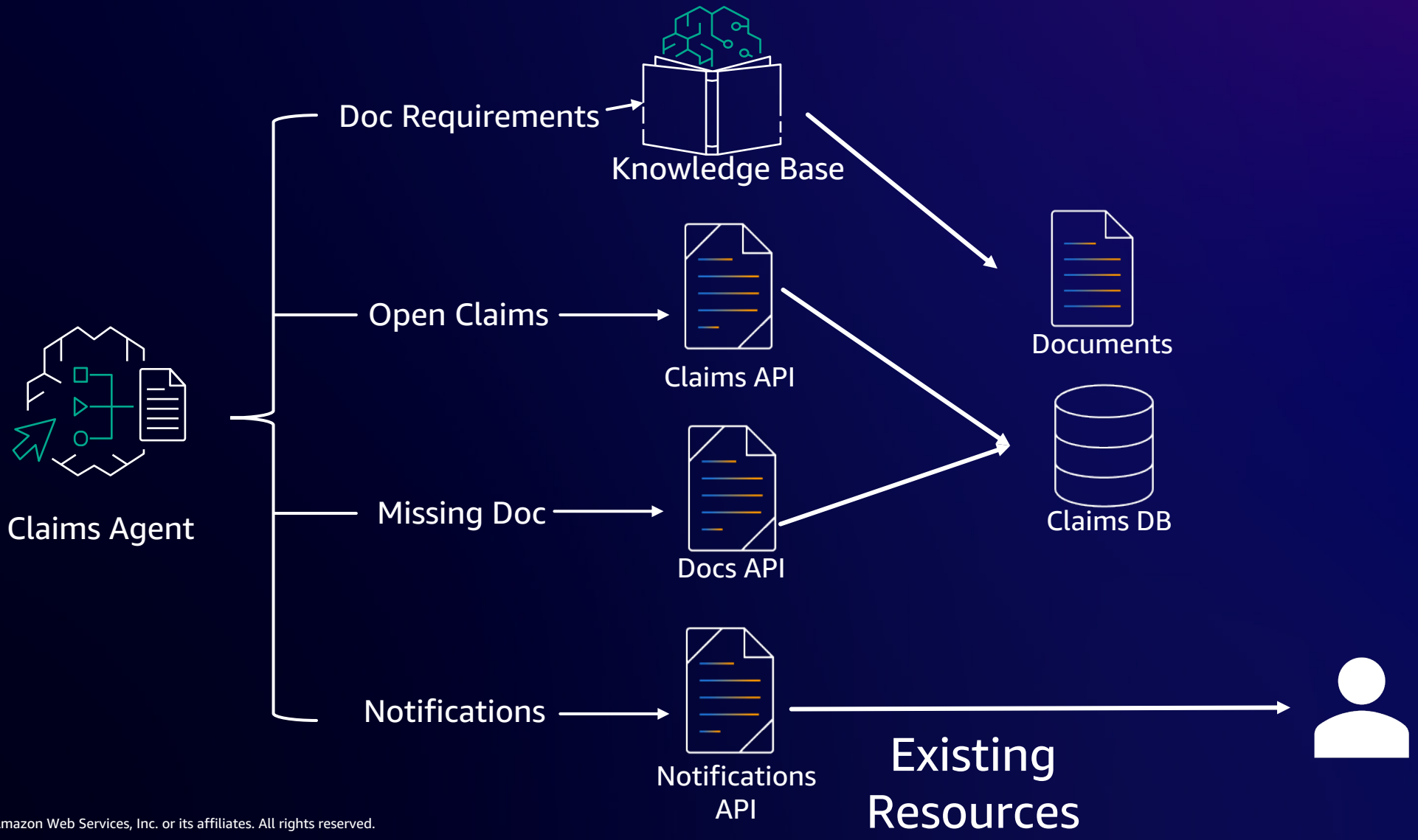
Execute action or search knowledge base

Observe results

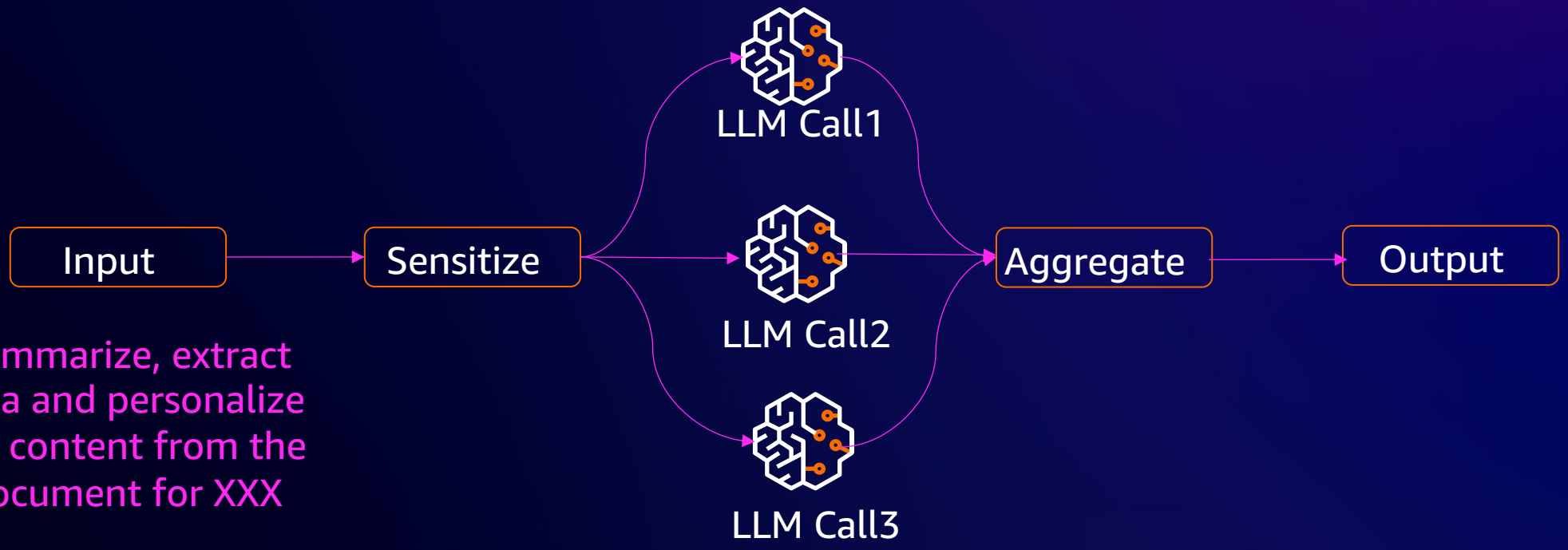
Think about next step

Until final answer

Agents build on existing enterprise resources



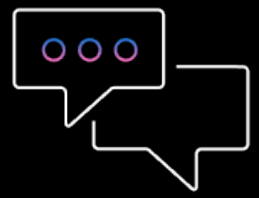
AI workflow (non-agentic AI) Example



Agentic AI vs Non-Agentic AI Applications

Feature/Aspect	Agentic AI Applications	Non-Agentic AI Applications
Definition	autonomously performs tasks, makes decisions, and take actions to achieve goals.	provides insights, predictions, or recommendations without taking actions.
Autonomy Level	Moderate to High Operates independently with minimal human intervention.	Low. Requires user input or direction to provide outputs.
Interactivity	Interactive, often engaging in conversations, executing workflows, or adjusting actions based on context.	Typically non-interactive; outputs are static and only respond to input.
Decision-Making	Uses decision-making frameworks, and goals to make real-time decisions.	Limited decision-making
User Control	User sets parameters, but agent operates independently within these boundaries.	User is in control, interpreting AI outputs and making decisions based on them.
Implementation Challenge	More complex to build due to need for governance, ongoing adaptation, autonomy, and interaction capabilities.	Easier to implement as outputs are typically straightforward and predefined

Types of Agents



Real-Time Agents

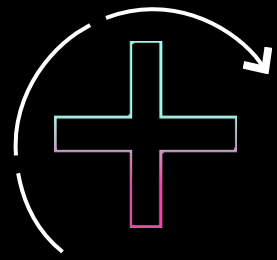
Real-time (humans waiting), synchronous, latency sensitive use cases.

Common instances - customer service and chat-based Q&A.

Task-Based Agents

Agents that operate on tasks or triggered by events.

Common instances - research reports, financial analysis, recommendations.



Goal-Seeking Agents

Agents that achieve a specific goal. Can be short or long-duration.

Common instances - code migration, code generation.

Implementing Agentic AI on AWS

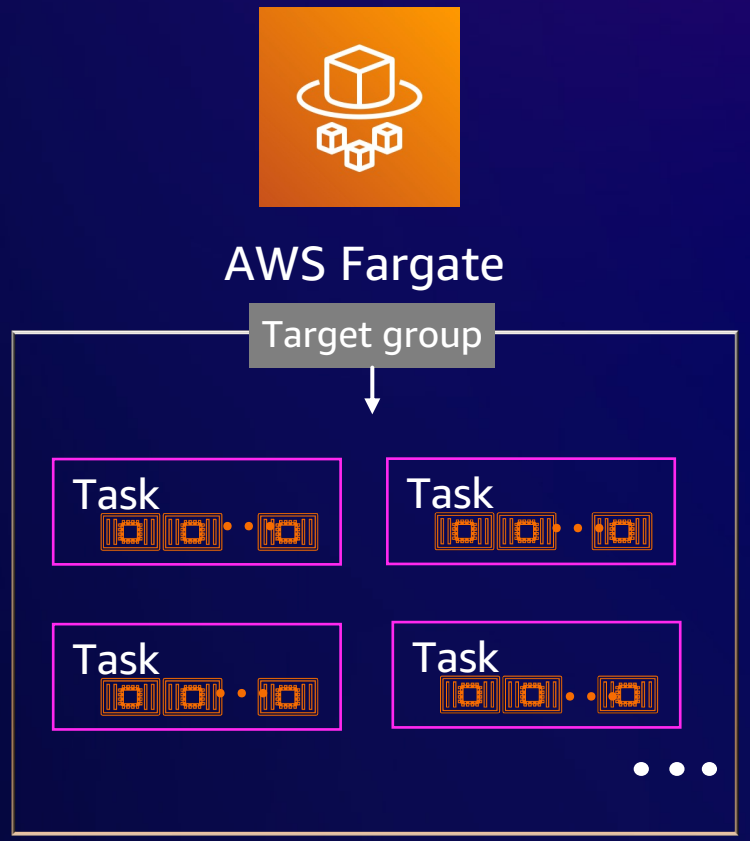
Different approaches to implement Agentic AI

- Containerize your favorite Agent framework
- Amazon Bedrock Agents
- Use Step Functions as agent

1. Containerize your agent

Building Agents with Open-source

- Build agents with LangGraph, LlamaIndex using Bedrock Converse API.
- Deploy, scale and manage
- Strands Agents



Building Agents with Strands Agents SDK

- Strands Agents is a simple-to-use, code-first framework for building agents.
- Models in Amazon Bedrock, OpenAI and other models through LiteLLM, Llama API…
- Built-in metrics, traces and logs
- Deploy in Lambda, ECS, EKS etc

```python
from strands import Agent

# Create an agent with default settings
agent = Agent()

# Ask the agent a question
agent("Tell me about agentic AI")
```

2. Amazon Bedrock Agents

Amazon Bedrock Agents

ENABLE GENERATIVE AI APPLICATIONS TO EXECUTE MULTISTEP TASKS USING COMPANY SYSTEMS AND DATA SOURCES

1

SELECT YOUR FM

2

PROVIDE BASIC INSTRUCTIONS

3

SELECT RELEVANT DATA SOURCES

4

SPECIFY AVAILABLE ACTIONS

Benefits

Automates orchestration of multi-step tasks

Simplifies building and deploying AI assistants

Provides secure access to enterprise data and APIs

Let’s you choose implementation languages

Provides fully managed infrastructure

Action Groups

Define Action Group

- Overview
- Relevancy

Define API Schema

- Operation name,
- Input parameters,
- Data types,
- Response details

Define Lambda functions

- Business logic
- Choice of programming language (Python, C#, JavaScript, Java, …)

Agents can be deployed and invoked from any app

Building and testing agents

Agent builder

Agent console or SDK

Agent draft

Using production agents

Agent user

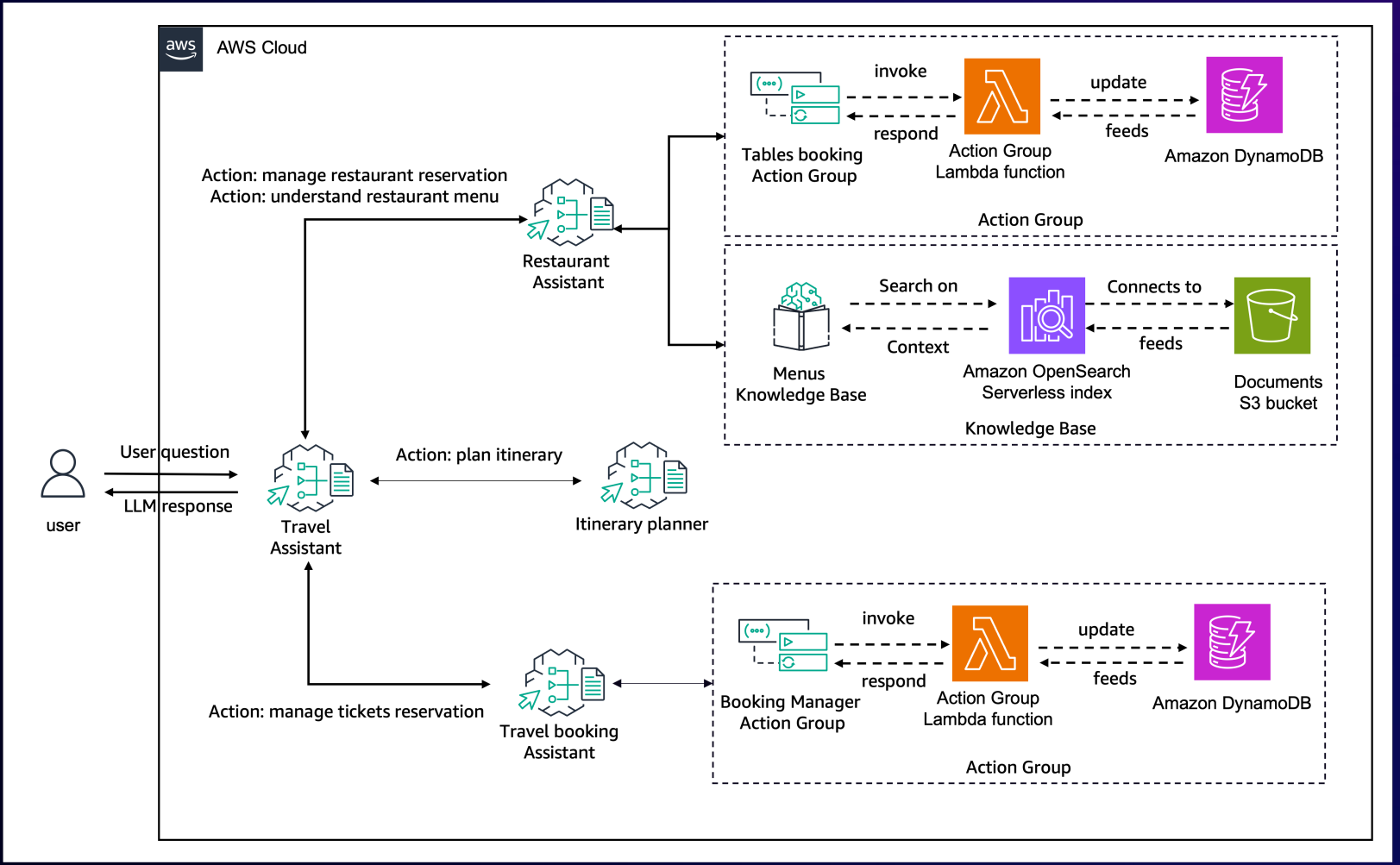
Customer app

Event

Deployed Agent

To deploy an agent, you create a new Alias, and optionally a new Version

Example on Bedrock/AWS



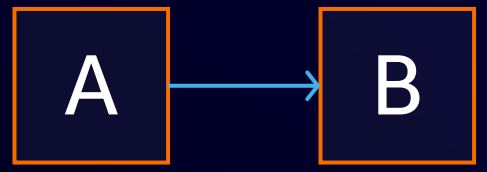
"By using Agents for Amazon Bedrock we turned a labor-intensive, 80-hour manual process into a few minutes, freeing valuable resources to work on higher value tasks such as system and business process design."

Shawn Swaner, CTO of Athene Holdings

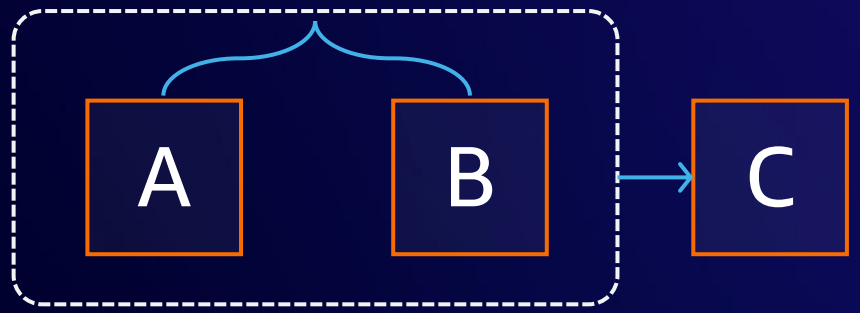
3. Use Step Functions

AWS Step Functions is a Serverless orchestrator

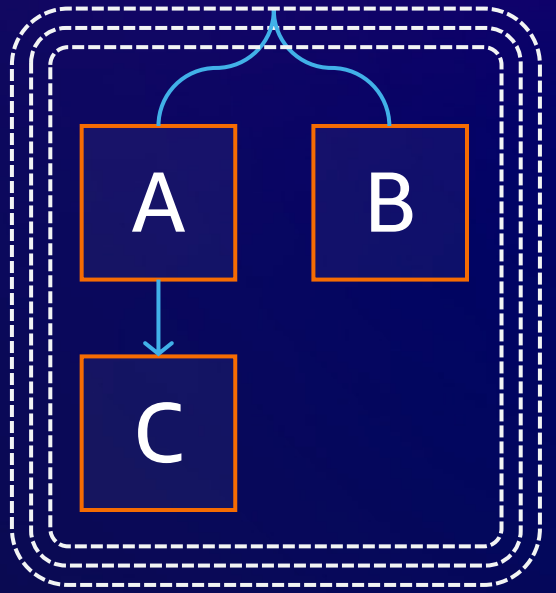
"I want to sequence tasks"



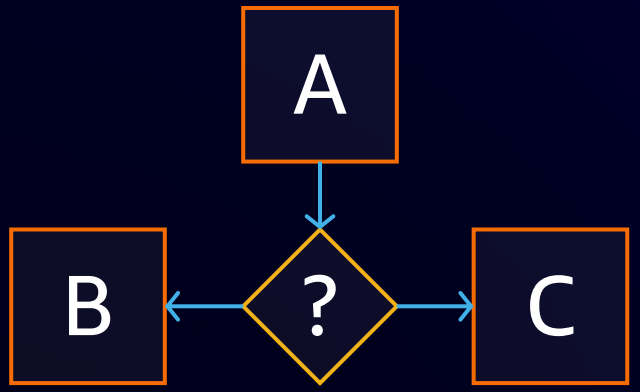
"I want to run tasks in parallel"



"I want concurrent and iterative tasks"



"I want to select tasks based on data"

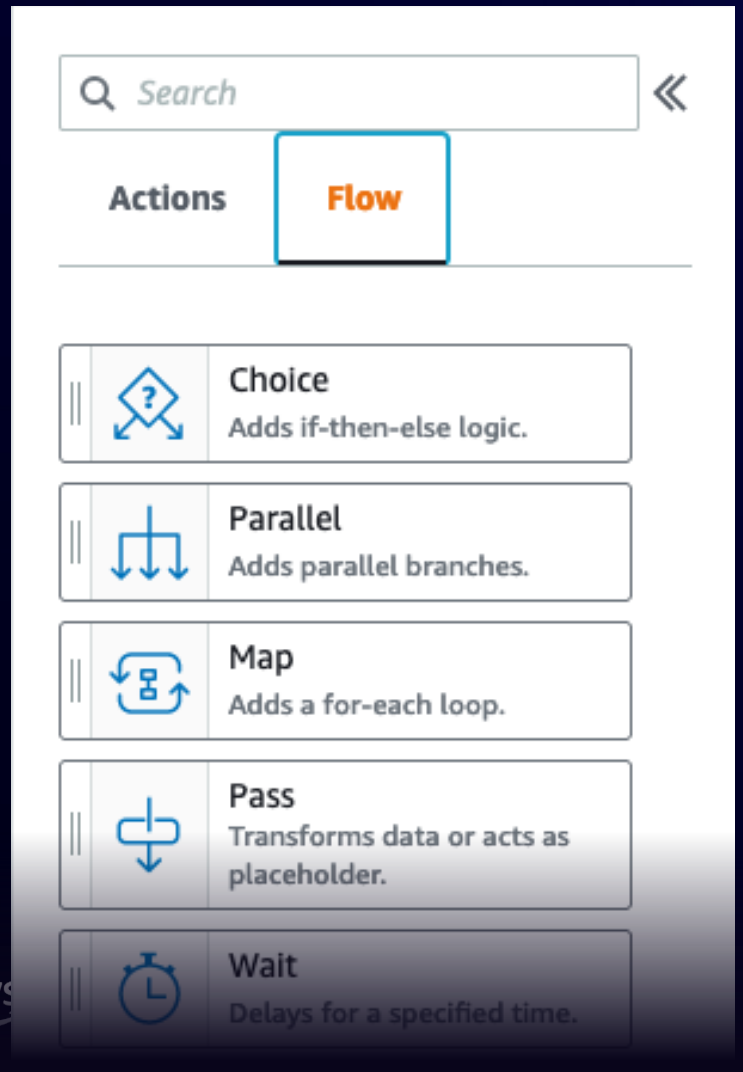


"I want wait (re)try/catch"

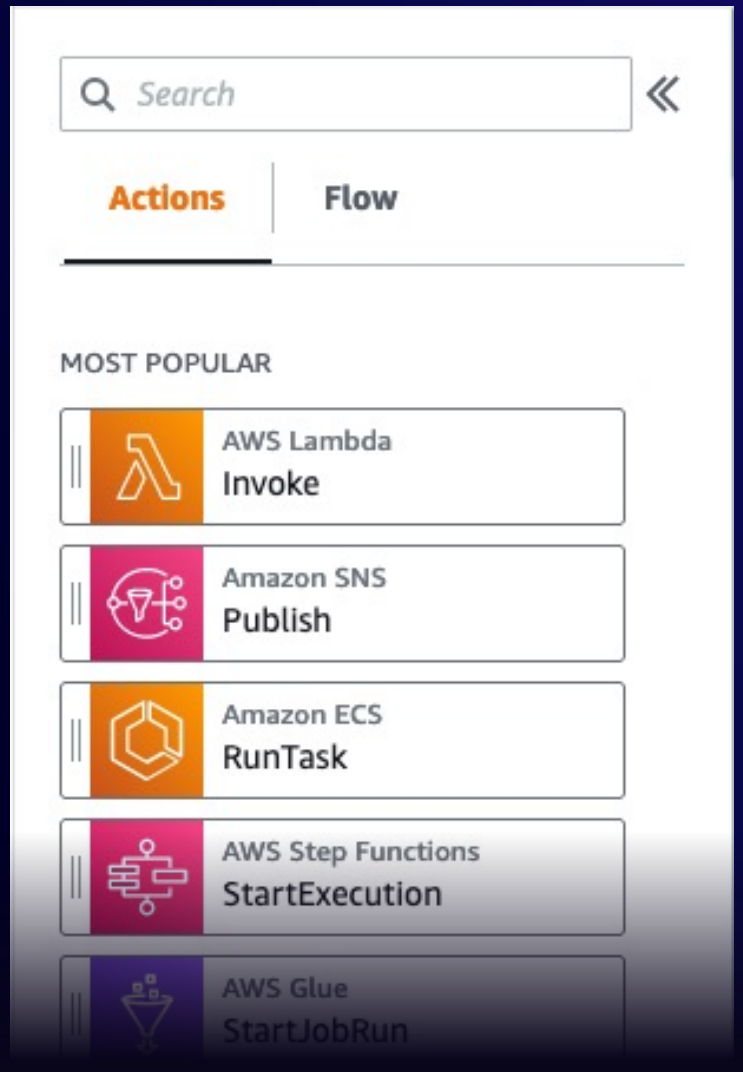
"I want Human review"

Workflow Studio | Integrations sidebar

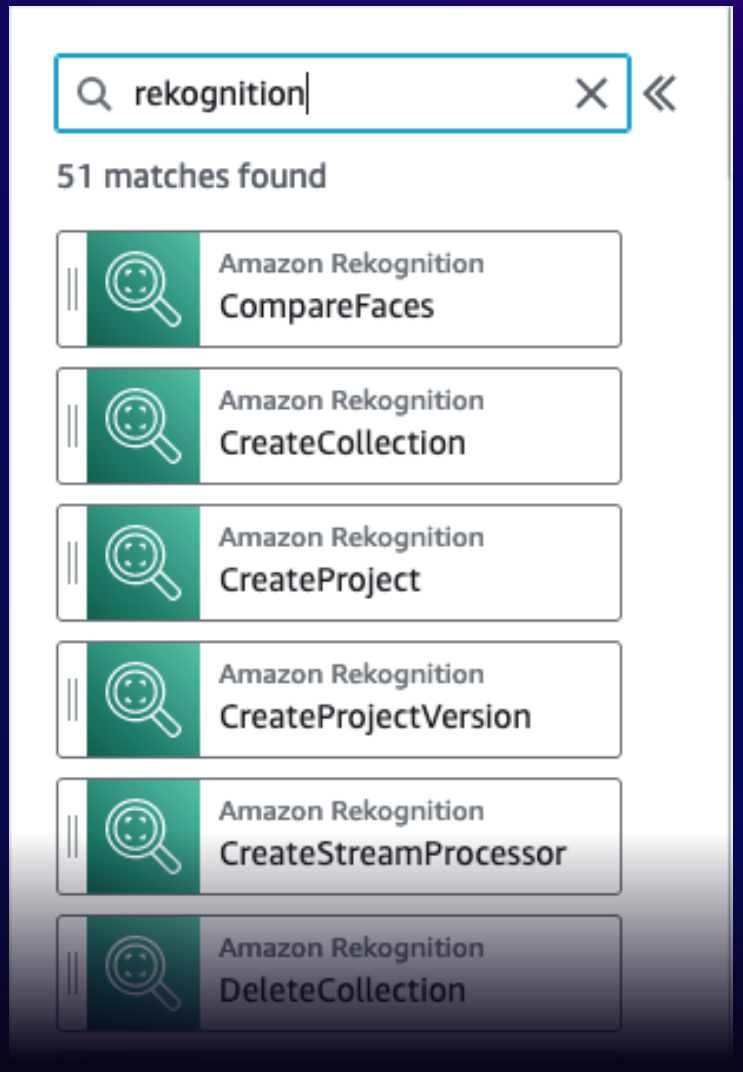
Flow States



Optimized Integrations



AWS SDK Integrations (11,000+)



How does Step Functions replace agents?

AI Agent Characteristics – Planning

Planning

Send a reminder to policy holders with missing docs; include doc requirements

Decomposition:
To answer this question, I will:

1. Get open claims
2. Get missing documents for each open claim
3. Get requirements for each missing document
4. Send reminders for each missing claim

AI Agent Characteristics – Tool Use

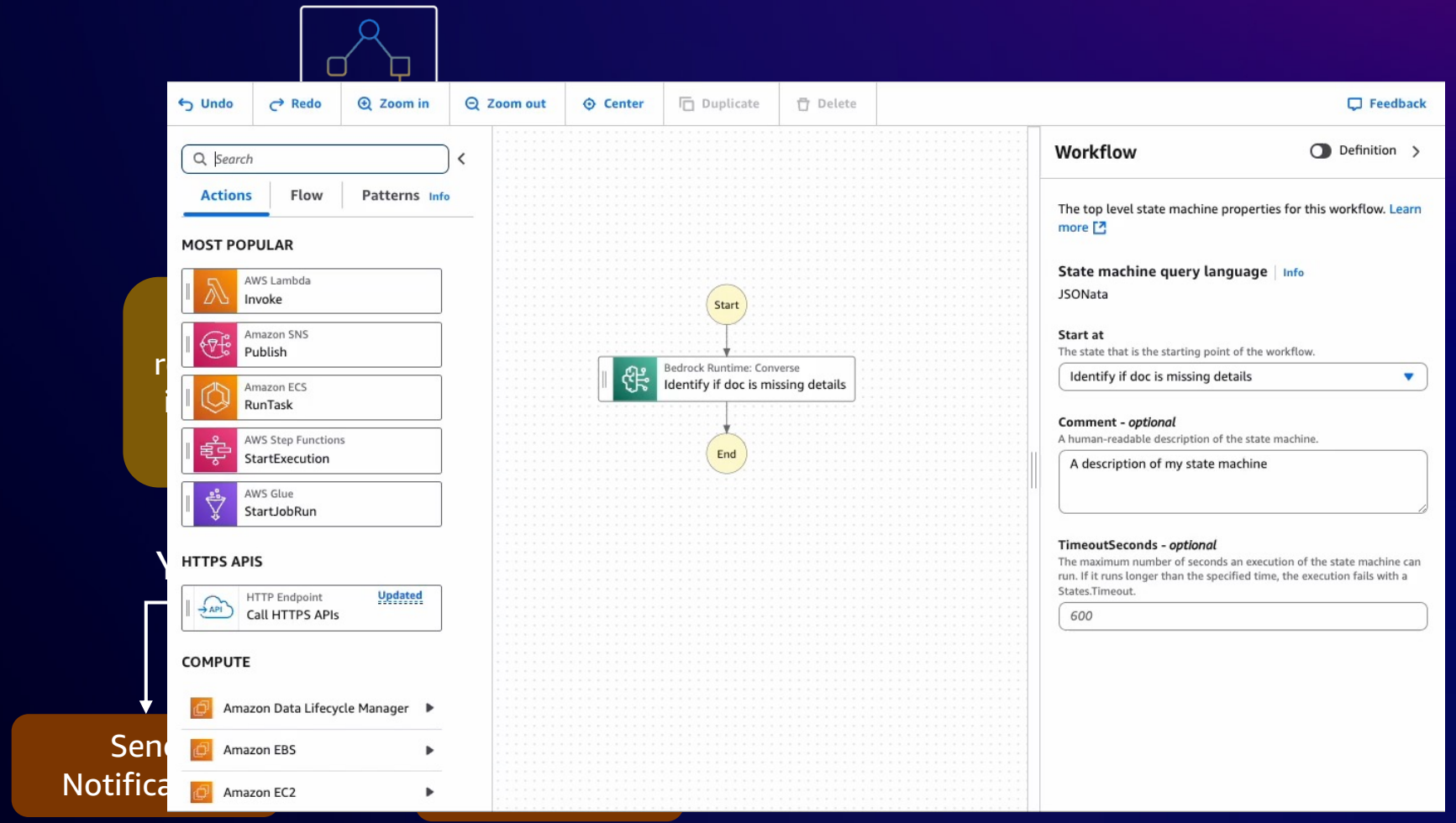
Tool Use

Send a reminder to policy holders with missing docs; include doc requirements

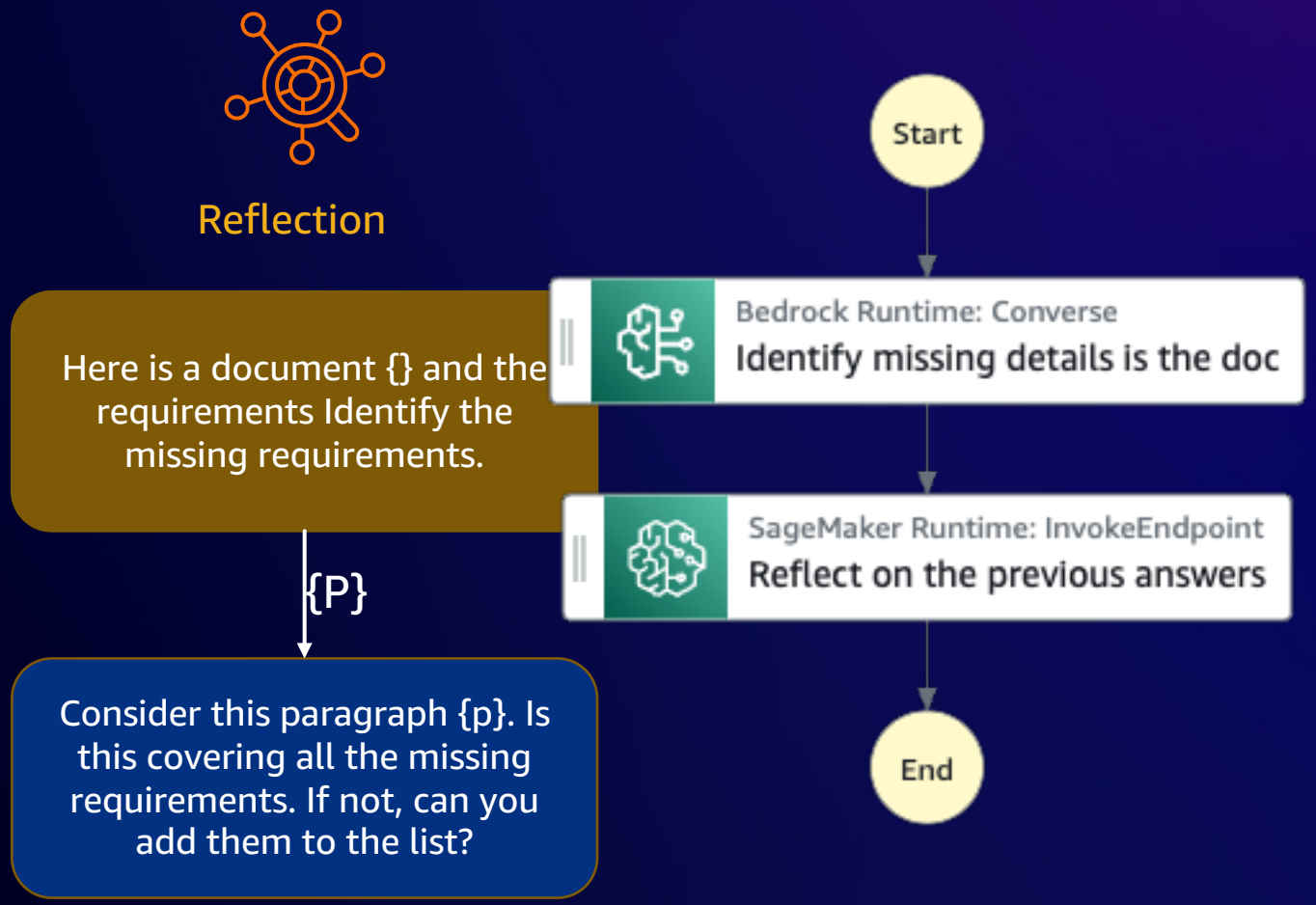
{getOpenClaims}

Invoke the getOpenClaims API

AI Agent Characteristics - Routing



AI Agent Characteristics - Reflection

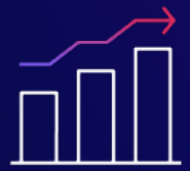


Benefits - Step Functions as agent

Freedom to choose the models

Modular Architecture

1. Scaling tools and agents independently
2. Nested workflows for reusability



Scale & Performance

1. Truly Serverless with high scalability
2. Asynchrony
3. Multi-agent and human-in-the-loop

Visual experience with development and operation

Considerations - Step Functions as agent

- Larger payload (>256KB) needs S3 or DynamoDB
- Rate control requires custom implementation
- External memory such as DynamoDB is needed for checkpointing across agents
- Limited local development and debuggability

Defining Tool Use

Prompt

Send a reminder to policy holders with missing docs; include doc requirements

Model Id: X

Tools

GetOpenClaims

Description: Provides the list of open claims

Schema:

```
"input_schema": {
   "type": "object",
    "properties": {
     ......
    }
  }
}
```

CompilePolicyIDs MissingDocs

Description: Gives the list of missing documents for policy IDs

Schema: . . .

SendReminders

Description: Sends emails and reminders given recipient and message

Schema: . . .

knowledge base

Description: Policy document requirements

Defining Tool Use

Prompt

Send a reminder to policy holders with missing docs; include doc requirements

Model Id: X

Tools

GetOpenClaims

Description: Provides the list of open claims

Schema:

```
"input_schema": {
   "type": "object",
    "properties": {
     ......
    }
  }
}
```

CompilePolicyIDs MissingDocs

Description: Gives the list of missing documents for policy IDs

Schema: . . .

SendReminders

Description: Sends emails and reminders given recipient and message

Schema: . . .

knowledge base

Description: Policy document requirements

Tool Execution

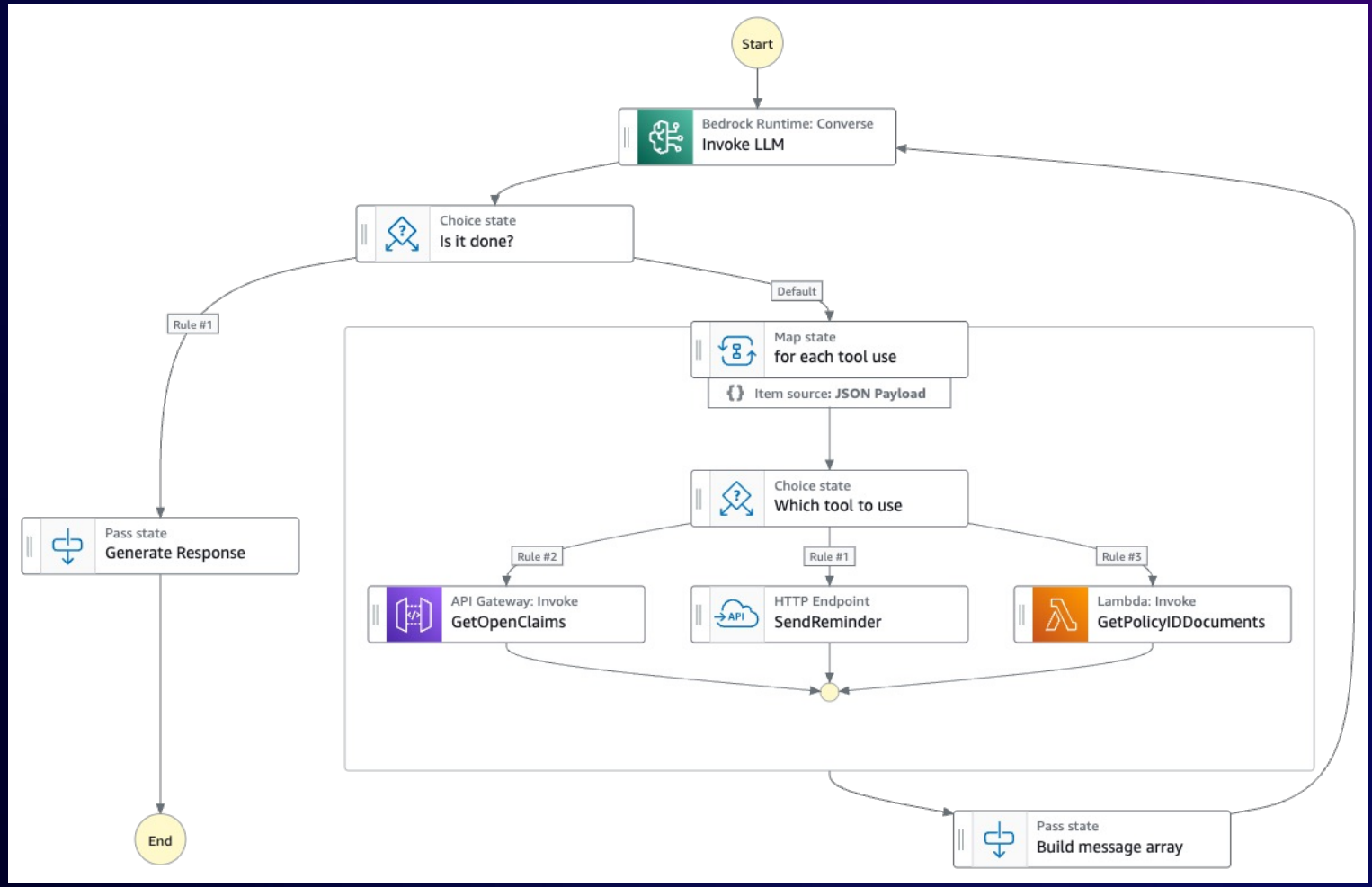
PolicyIDMissingDocs

GetOpenClaims

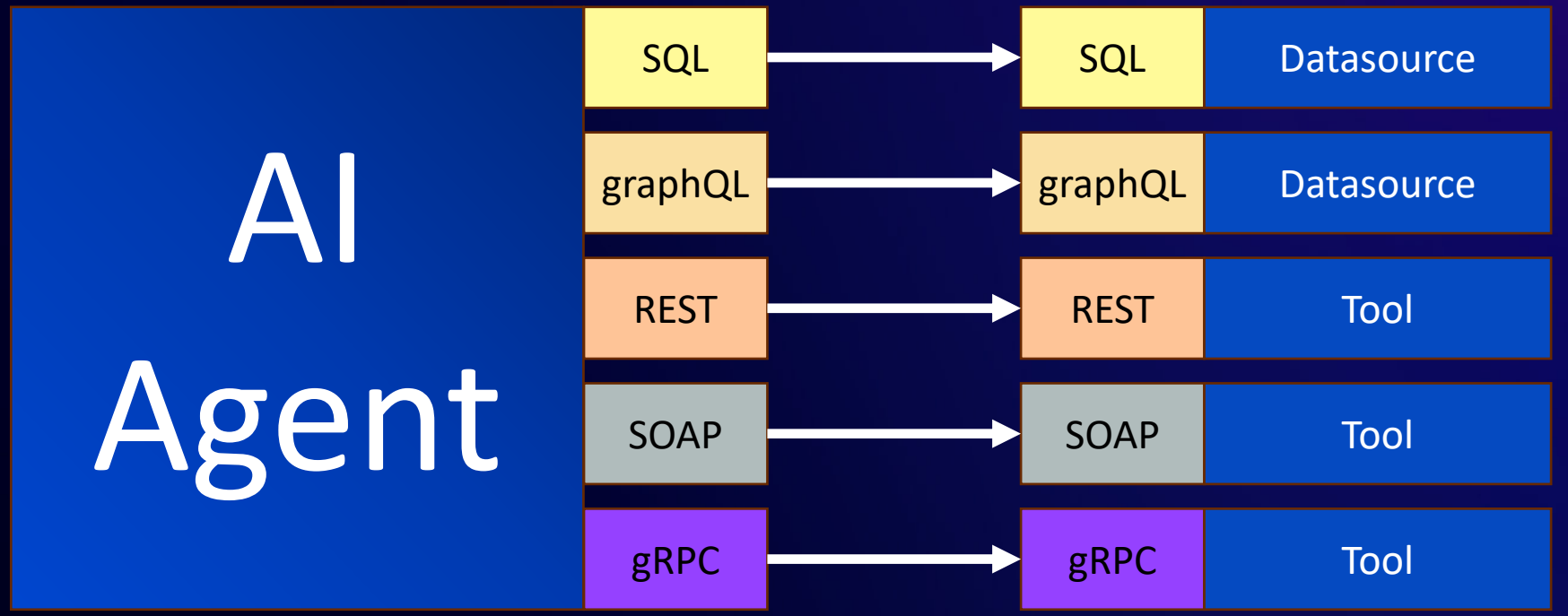
SendReminders

Step Functions with Tool Use

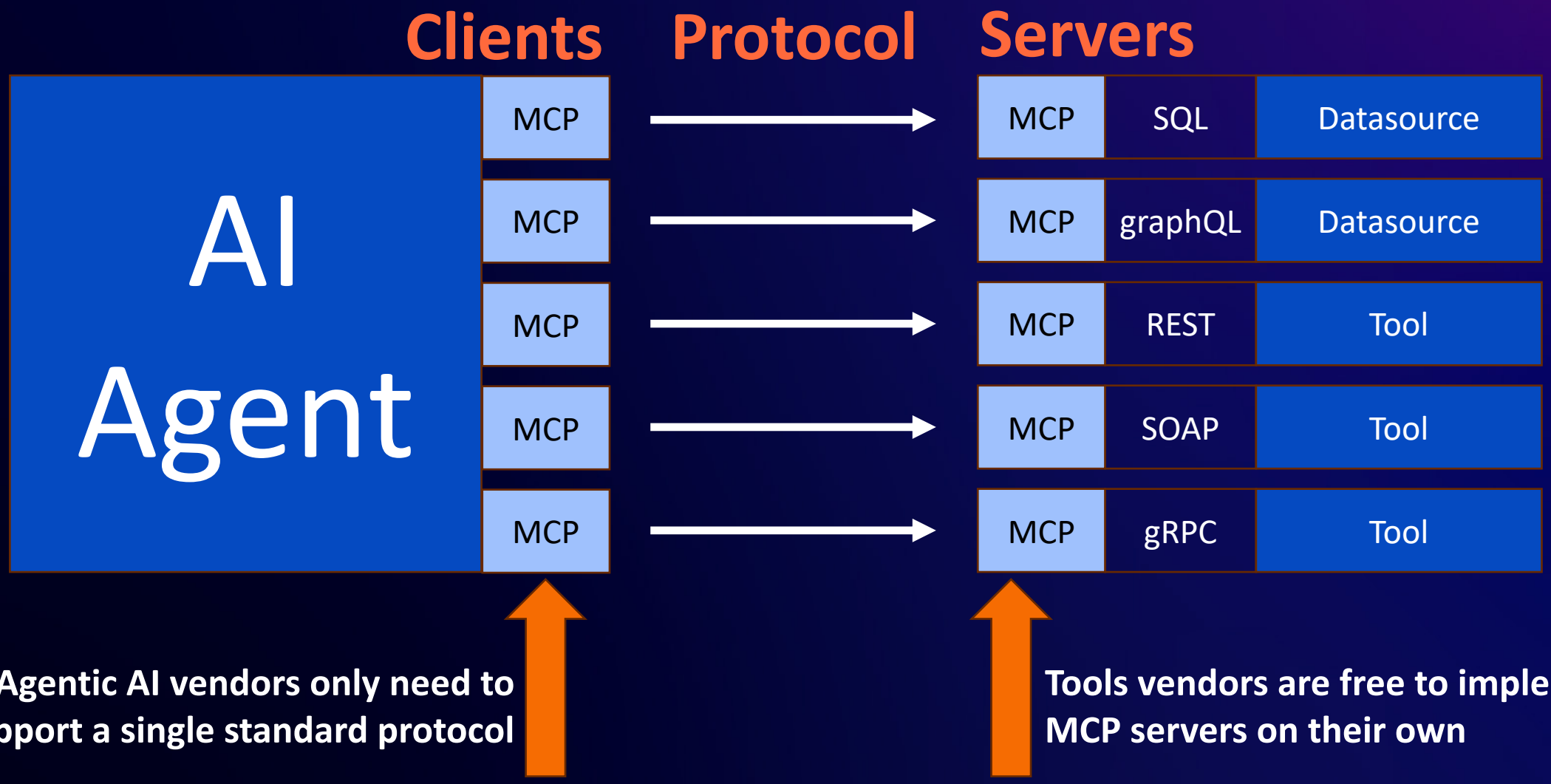
- Invoke LLM to plan and identify tools
- Check if this is no more tools
- If tools, invoke tool
- Keep iterating until no tool



Communicating with multiple tools and resources

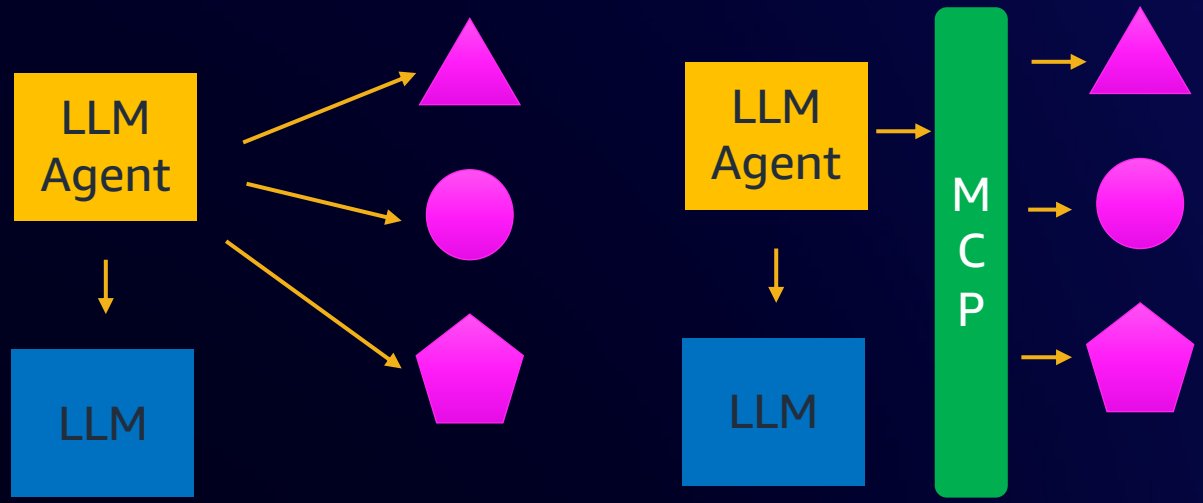


Communicating with multiple tools and resources



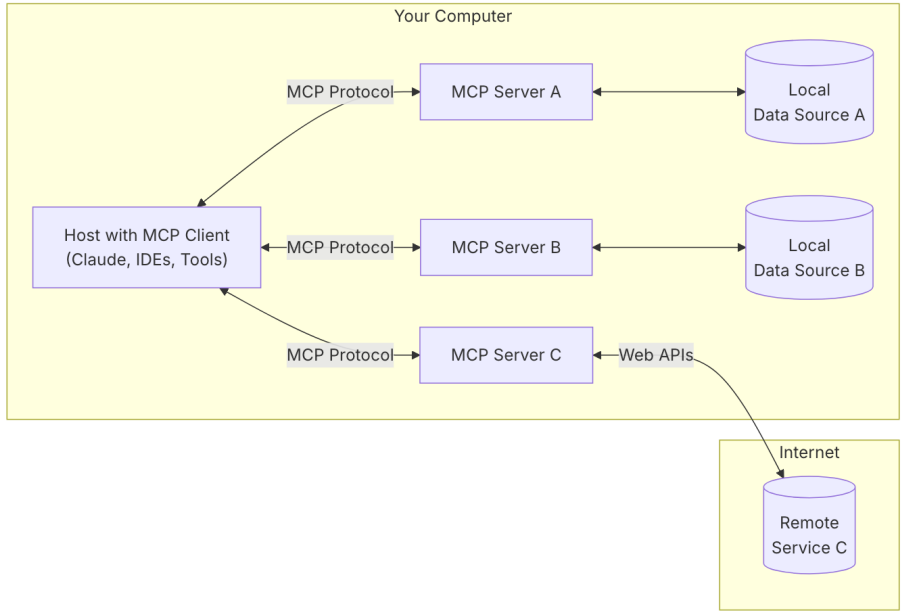
MCP (Model Context Protocol)

- Coined by Anthropic
- Standardization around Agentic AI
- REST for Microservices where MCP is for Agentic AI



General architecture

At its core, MCP follows a client-server architecture where a host application can connect to multiple servers:



- **MCP Hosts**: Programs like Claude Desktop, IDEs, or AI tools that want to access data through MCP
- **MCP Clients**: Protocol clients that maintain 1:1 connections with servers
- **MCP Servers**: Lightweight programs that each expose specific capabilities through the standardized Model Context Protocol
- **Local Data Sources**: Your computer's files, databases, and services that MCP servers can securely access
- **Remote Services**: External systems available over the internet (e.g., through APIs) that MCP servers can connect to

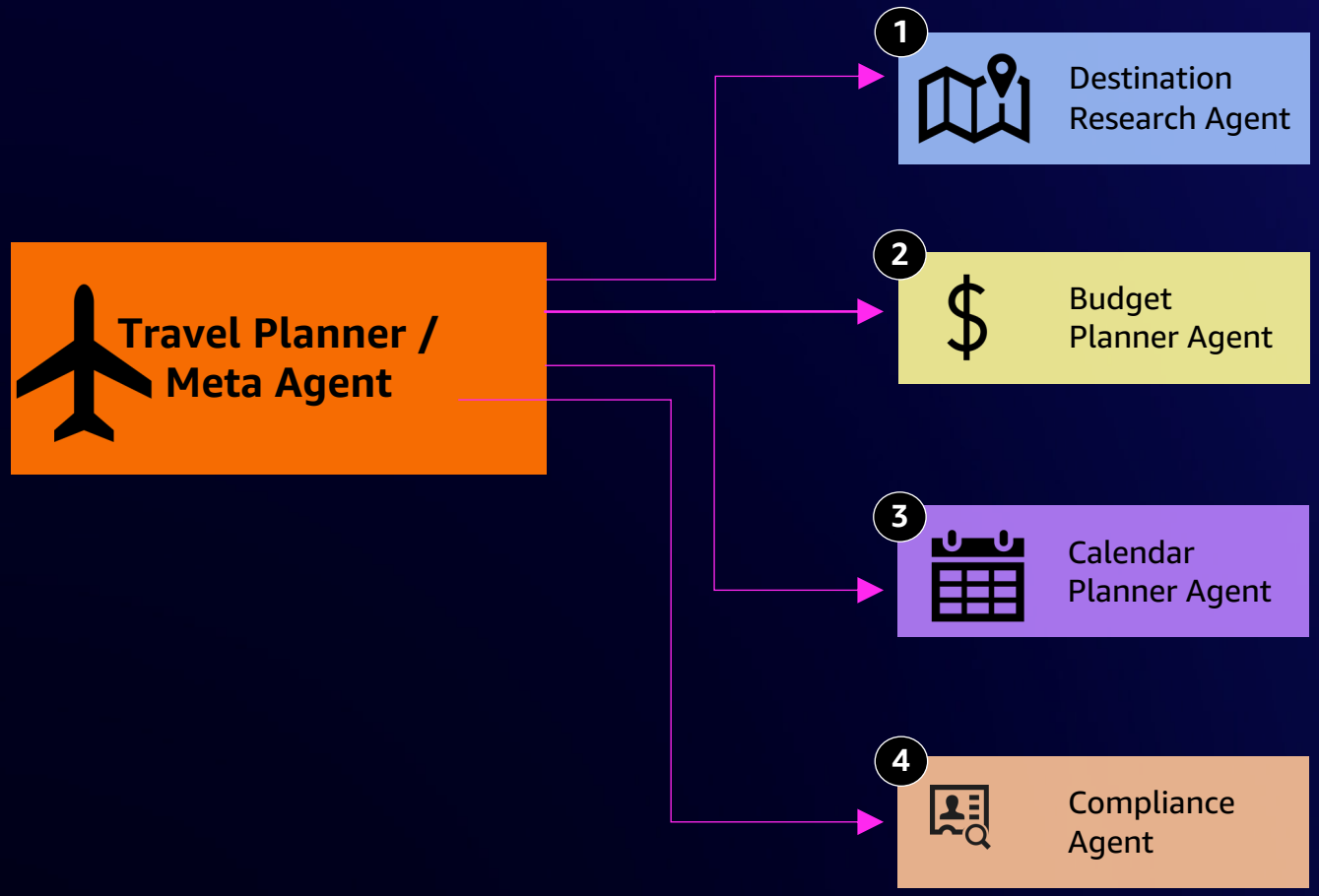
Which Agent architecture works for me?

Which agent works for me?

Criteria	LangGraph	Step Functions as agents	Bedrock Agents
Architecture	Graph-based architecture	Workflow based architecture	LLM-based architecture
Integrations and tool support	Part of LangChain framework, extensive tool support options	11K AWS APIs, Public and private APIs and on-premise systems	Integration with external APIs using Actions, Action Groups powered by AWS Lambda
Memory and Context Management	Long-term, short-term and contextual memory with time travel features for debugging	State management and DynamoDB can be used as memory	Fully managed memory management with APIs, provided control and flexibility
Other Features	Excels in stateful applications	Fit for general purpose AI workflows . static agentic workflows, Agent orchestrations	Fit for single agent workflows
Ease Of Use	Provide granular control, has steep learning curve	Intuitive UI builder, Built-in retry, error handling and observability	Simple and opinionated agentic workflows
Operational responsibilities	Need to manage scale and deployment externally	Serverless	Serverless

Common patterns

Agents can be many

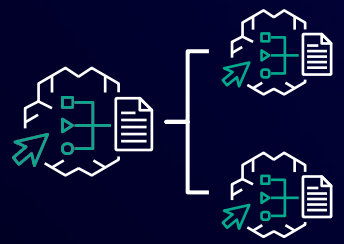


Travel Planner (Meta Agent): Understands the user goal, breaks down the tasks and select best sub agent for each task.

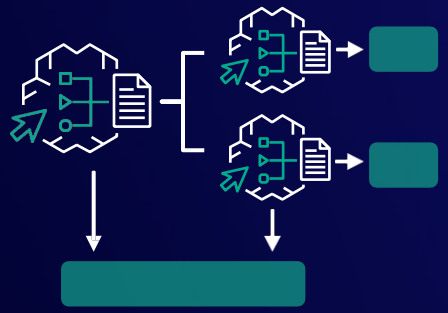
- **Destination Research Sub Agent:** compiles a list of top destinations based on the goal.
- **Budget Planner Agent:** compiles a list of travel and accommodation options for each destination within budget constraint.
- **Calendar Planner Agent:** determines the best times of the year to visit each destination based on prior plans and seasonality.
- **Compliance Agent** reviews all the collected information, cross-validating the data to ensure its reliability and accuracy.
- **Meta Agent** compiles and formats the final comprehensive travel guides.

Agentic Patterns

Single Agents



Supervised Agents



Hierarchical Agents

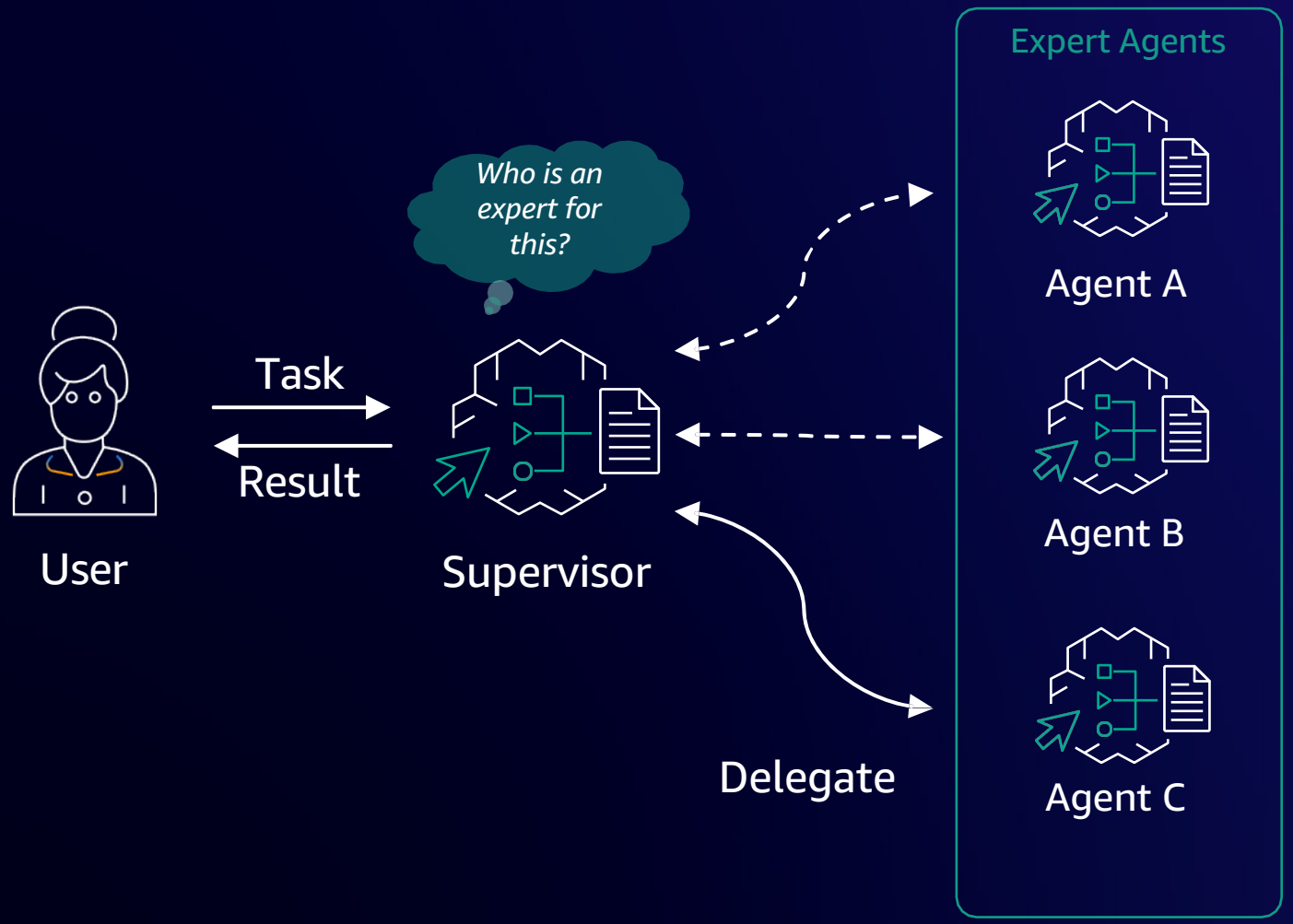
Async Agents

Autonomous Agents

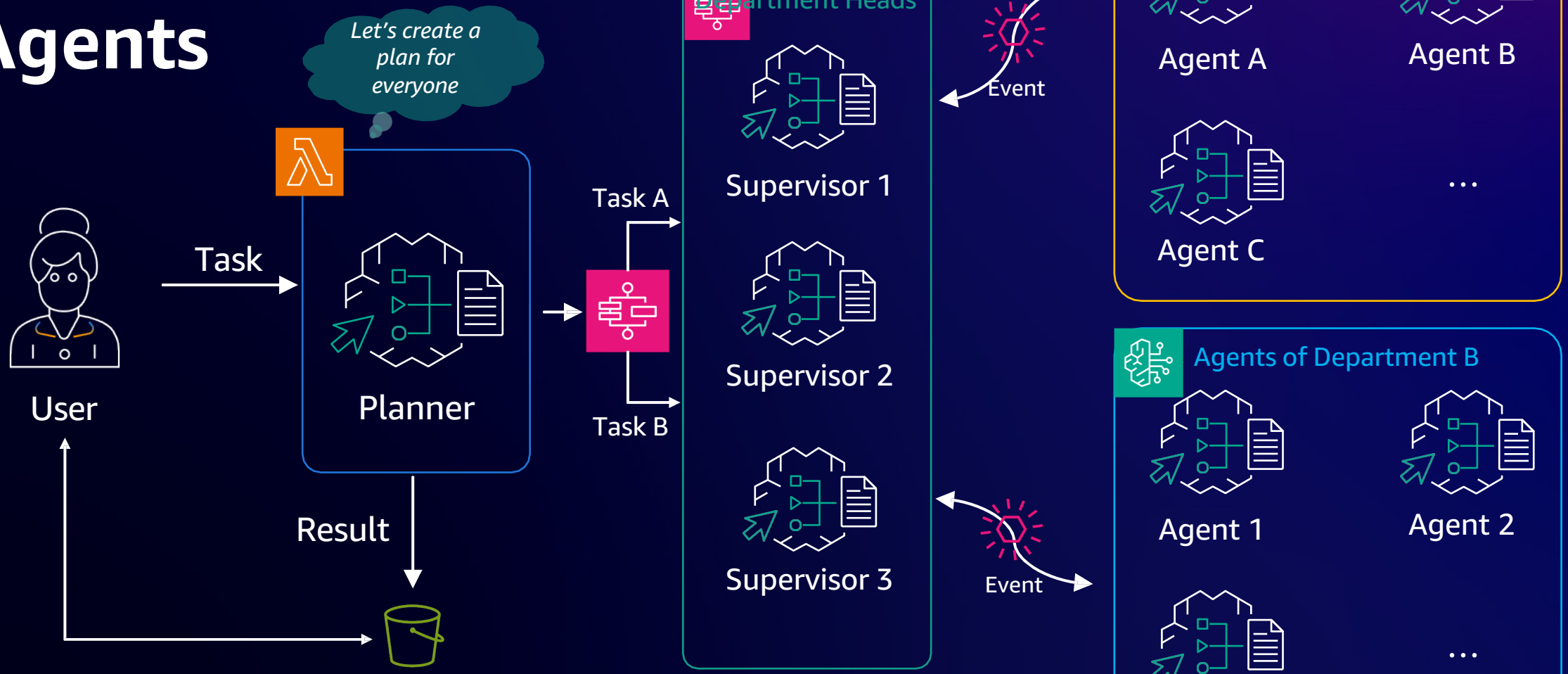
Simple

Complex

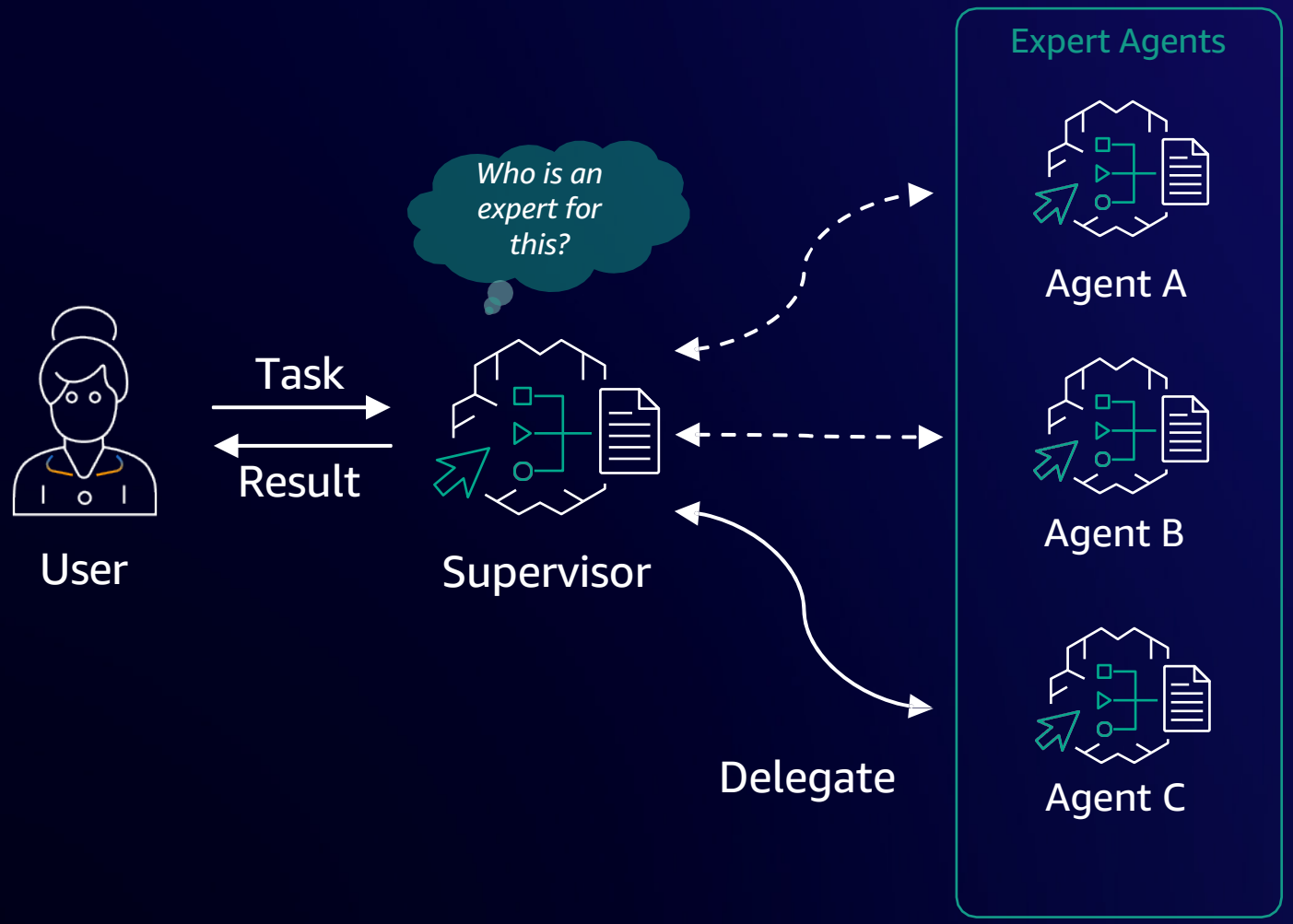
Multi-Agent Orchestration with Supervisor



Async Agents



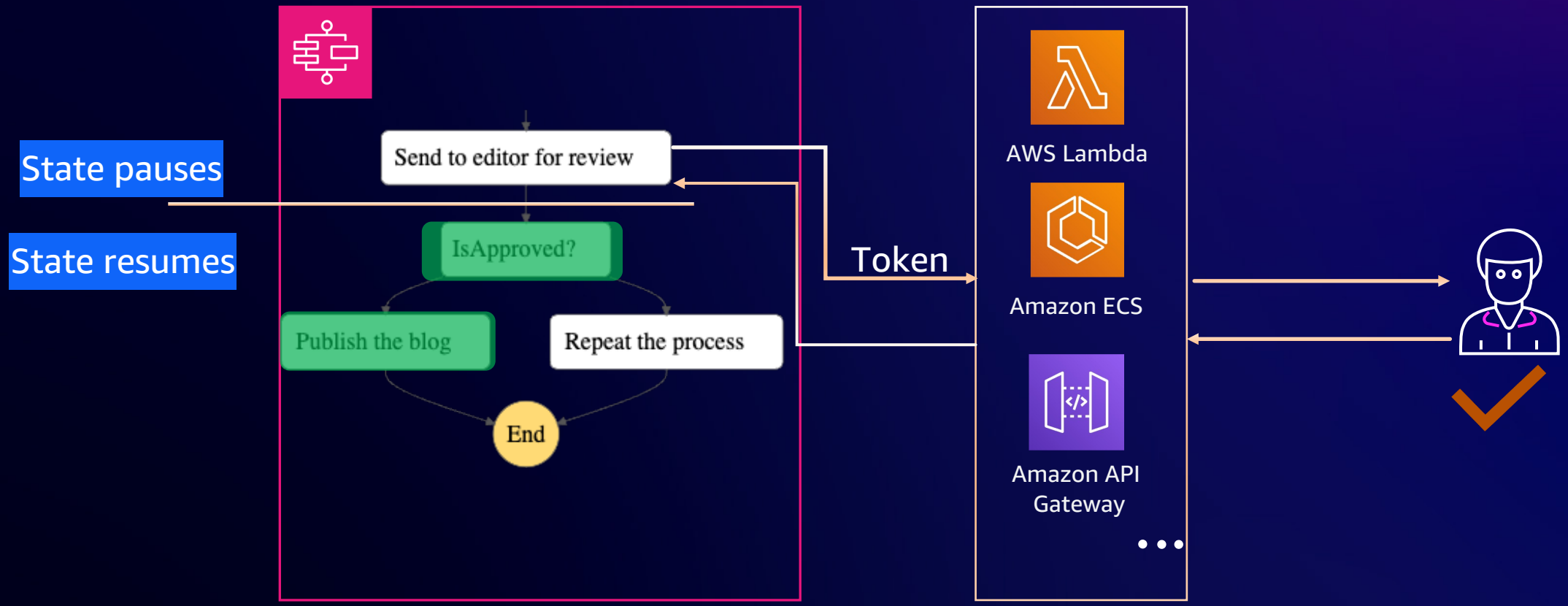
Human in the loop



Human approval

- Safety and accuracy is critical
- Automated validation fails
- Data preparation as well as inferencing

Human-in-the-loop pattern with Step Functions



Best practices

Agent Evaluation

- Build context aware evaluation metrics
- Availability of ground truth and objectives for agent evaluation is manual effort, expensive and error prone
- Lack of standard accepted framework or set of best practices for evaluating AI agents may demand its own custom evaluation criteria, tools, and benchmarks.

Operations

- **Resource Management**: Monitor and control resource consumption to avoid inefficiencies and resource exhaustion due to agent over-proliferation.
- **Performance Scalability**: As agents increase, manage CPU, memory, and bandwidth to prevent contention and bottlenecks.
- **Communication Overhead**: Reduce network congestion and delays by optimizing agent communication to maintain responsiveness and avoid message loss.

Final thoughts on agents

- Agents are not drop-in upgrade for every use case
- Keep it simple
- Models are only as good as the context provided to them

Start your Agentic AI journey today

AWS Community

Step by step tutorial

Hands-on workshop

Survey

https://pulse.aws/survey/TUE5KIWM

Workshop

https://catalog.us-east-1.prod.workshops.aws/join?access-code=4fb0-0bac80-56

Thank you

/vamsivikash/